REVUE FRANÇAISE D'AUTOMATIQUE, D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE. RECHERCHE OPÉRATIONNELLE

J. CARLIER

P. CHRÉTIENNE

Un domaine très ouvert : les problèmes d'ordonnancement

Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 16, n° 3 (1982), p. 175-217.

<http://www.numdam.org/item?id=RO_1982__16_3_175_0>

R.A.I.R.O. Recherche opérationnelle/Operations Research
(vol. 16, n° 3, août 1982, p. 175 à 217)

UN DOMAINE TRÈS OUVERT : LES PROBLÈMES D'ORDONNANCEMENT (*)

par J. CARLIER et P. CHRÉTIENNE (1)

Résumé. — *Le but de ce papier est de présenter des méthodes pour traiter les problèmes d'ordonnancement.*

Beaucoup de problèmes économiques sont ou contiennent des problèmes d'ordonnancement; par exemple, les problèmes d'ateliers, les problèmes d'emploi du temps, et les problèmes de suivi de projet. Aussi ces derniers ont suscité depuis une dizaine d'années de nombreuses recherches théoriques et pratiques.

Nous rappelons d'abord les méthodes classiques de l'ordonnancement : méthode C.P.M./P.E.R.T., méthode des potentiels, graphique de Gantt et méthodes sérielles.

Ensuite, nous présentons les principales caractéristiques de ces problèmes : tâches, ressources, machines, contraintes potentielles et fonction économique.

Enfin, nous présentons quelques résultats récents sur les problèmes à une machine, les problèmes à m machines et les problèmes d'atelier. Notre but est de montrer les différentes façons d'aborder ces problèmes : méthodes exactes et approchées, calcul de bornes, relaxation...

Nous concluons en remarquant que le domaine de recherche reste très ouvert et qu'il reste, par exemple, à relier certains résultats théoriques aux applications.

Mots clés : Tâche; ressource; ordonnancement; algorithme; complexité;

Abstract. — *The aim of this paper is to present a general survey of scheduling problems and relevant methods.*

In many economic situations, we have to face scheduling problems, for instance, in job-shop and in building. So they have been studied intensively for a decade from theoretical and practical point of view.

First we present classical scheduling problems: C.P.M./P.E.R.T. methods, Gantt charts and list techniques.

Then we introduce the main features of scheduling problems: tasks, machines, resources, potential inequalities and objective functions.

The last part is devoted to some new results dealing with one-machine problems, m-machines problems and job-shop scheduling.

We don't intend to be exhaustive in the field of scheduling theory; we would rather like to show some of the ways to take up these problems: exact or approximate methods, bounds derivation,...

We conclude that scheduling is a widely opened field and that much work is still to be done, for example developping the use of theoretical results in practical applications.

Keywords : Task; resource; schedule; algorithm; complexity.

(*) Reçu décembre 1980.

(1) Institut de Programmation. Université Paris-VI, Tour 55-65, 4, place Jussieu, 75230 Paris Cedex 05.

R.A.I.R.O. Recherche opérationnelle/Operations Research, 0399-0559/1982/175/$ 5.00

1. INTRODUCTION

Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution.

Les problèmes d'ordonnancement apparaissent dans tous les domaines de l'économie : l'informatique (tâches : jobs; ressources : processeurs ou mémoire...), la construction (suivi de projet), l'industrie (problèmes d'ateliers, gestion de production), l'administration (emplois du temps).

La théorie de l'ordonnancement traite de modèles mathématiques mais aussi de la modélisation de situations réelles fort complexes; aussi le développement de modèles utiles ne peut être que le fruit de contacts entre la théorie et la pratique.

A ce jour seules sont appliquées industriellement les méthodes simples : méthodes C.P.M./P.E.R.T., méthode des potentiels M.P.M., méthodes sérielles, de lissage, etc... (les deux dernières visant surtout la répartition des ressources). Mais elles ne permettent de traiter rigoureusement que les modèles les plus simples.

Aussi les modèles plus élaborés ont suscité depuis une dizaine d'années une recherche accrue (Baker [2], Coffman [12], Conway *et al.* [13], Graham *et al.* [20], Lenstra [30], Rinnooy Kan [35], Roy [37], Weglarz [44]) dont nous présentons quelques résultats.

Ces techniques sont peu connues car très récentes. Après avoir décrit la structure d'un problème d'ordonnancement, nous rappelons les méthodes appliquées; puis nous présentons les méthodes nouvelles tenant compte des contraintes de moyens.

2. DESCRIPTION DES PROBLÈMES D'ORDONNANCEMENT (Roy *et al.*, [38]) (²)

Ordonnancer c'est programmer dans le temps l'exécution d'une réalisation décomposable en tâches, en attribuant des ressources à ces tâches et en fixant en particulier leurs dates de début d'exécution tout en respectant des contraintes données.

Les différentes données d'un problème d'ordonnancement sont les tâches, les contraintes potentielles, les ressources et la fonction économique.

Les tâches sont le dénominateur commun des problèmes d'ordonnancement; leurs définitions ne sont ni toujours immédiates ni toujours triviales. Leurs définitions ne sont pas immédiates dans les problèmes de bâtiment, car elles

(²) Un glossaire en fin d'article récapitule les différentes notations.

dépendent de la finesse du découpage : par exemple, dans la construction d'un immeuble, on pourra, par appartement, soit considérer une tâche menuiserie, soit des tâches : pose de la porte d'entrée, pose d'un placard... Leurs définitions ne sont pas triviales : dans des problèmes de transport, une tâche sera, par exemple, l'occupation d'une portion de voie par un train.

Ces tâches peuvent être liées par des contraintes « potentielles » ([3]); celles-ci englobent des contraintes de succession : par exemple, le gros œuvre suit les fondations, la pose des tuiles est postérieur à la pose de la charpente, etc. Elles englobent aussi des contraintes de localisation temporelle : la tâche (i) doit être achevée à telle date, ou, au contraire, son exécution ne doit pas commencer avant telle date.

Ces tâches requièrent pour être exécutées certaines ressources (Carlier [9], Erschler *et al.* [14]) : les tâches peuvent passer sur des machines, utiliser de la main d'œuvre, devoir être financées, etc. Ces ressources introduisent des contraintes « disjonctives » et des contraintes « cumulatives » : une contrainte « disjonctive » apparaîtra quand deux tâches par exemple parce qu'elles utilisent une même machine ne pourront pas s'exécuter simultanément; une contrainte « cumulative » apparaîtra par exemple pour un ensemble de quatre tâches de maçonnerie si trois maçons seulement sont disponibles : on ne pourra pas exécuter, à la fois, plus de trois de ces tâches sans que l'on puisse savoir à l'avance laquelle sera retardée.

Enfin, il faut programmer ces tâches de façon à optimiser un certain objectif qui sera suivant le cas, la minimisation de la durée totale (c'est le critère le plus fréquemment employé) ou le respect des dates de commande ou le lissage des courbes de main d'œuvre ou encore la minimisation d'un coût. En fait, il est souvent plus réaliste de considérer plusieurs critères.

Nous résumons dans la suite les différents concepts que nous avons évoqués dans cette description.

2.1. Les tâches

Nous utilisons ci-dessous les notations les plus couramment employées par les spécialistes d'ordonnancement.

Données relatives à une tâche

On notera t_i la date de début d'exécution de la tâche (i) et C_i sa date de fin d'exécution.

([3]) L'emploi du terme mathématique « potentiel » sera justifié plus bas.

L'exécution d'une tâche (i) de durée p_i (en anglais : « processing time ») peut commencer à la date r_i (date de début au plus tôt, « release date » en anglais) et doit être achevée à la date d_i (date de fin au plus tard, « due date » en anglais); on a donc :

$r_i \leqq t_i < C_i \leqq d_i$; d'autre part si la tâche (i) n'est pas interrompue : $C_i = t_i + p_i$.

Dans certains cas on aura à considérer un poids w_i pour cette tâche qui pourra être une pénalité si $C_i > d_i$, un coût de stock d'encours, etc.

Relation entre les tâches

Une relation d'antériorité entre les tâches apparaît souvent; cette relation sera traduite dans un graphe si l'on emploie la méthode C.P.M. ou celle des potentiels.

Préemption et non préemption

Dans certains cas, une tâche peut être exécutée par morceaux (par exemple, dans un système multiprogrammé où l'on peut morceler l'exécution d'un job); dans ce cas, on dit que la tâche est interruptible ou encore que la « préemption » est possible ([4]) : une tâche (i) est « préemptée » par une tâche (j) à l'époque t si (i) était en cours d'exécution avant t et employait une ressource qu'on alloue à la tâche j à t, provoquant ainsi l'interruption de (i).

Souvent la possibilité de préempter rend les problèmes plus faciles; ainsi, pourra-t-on autoriser la préemption pour calculer des bornes de la fonction objectif.

2.2. Les machines

Lorsqu'il y a des machines, elles peuvent être identiques ou différentes, dépendantes ou non les unes des autres. Certaines ne sont disponibles qu'à certaines périodes. Certaines tâches peuvent requérir des machines spécifiques.

2.3. Les ressources

Il peut y avoir d'autres ressources que les machines par exemple de la main d'œuvre, de l'argent, des fichiers (Informatique). On connaît en général les quantités de ressources (d'un type donné) disponibles en fonction du temps.

([4]) Le terme préemption est ici un anglicisme, il n'a rien à voir avec le sens habituel (droit de préemption).

2.4. La fonction économique

Les variables intervenant dans l'expression de la fonction économique sont :
- la date C_i de fin de la tâche (i);
- le retard $T_i = max\ (0,\ C_i - d_i)$ de la tâche (i);
- l'indicateur de retard U_i : $U_i = 0$ si $C_i \leqq d_i$ et $U_i = 1$, sinon.

Les fonctions économiques (critères) les plus utilisés font intervenir la durée totale, le délai d'exécution pondéré, les retards, et le coût des stocks d'encours.

Durée totale

La durée totale de l'ordonnancement, notée C_{max} est égale à la date d'achèvement de la tâche la plus tardive : $C_{max} = max\ C_i$.

Dans les méthodes C.P.M./P.E.R.T. et M.P.M., le critère adopté consiste à minimiser cette durée totale.

Respect des dates au plus tard

Dans beaucoup de problèmes, il faut respecter les délais, donc les dates au plus tard d_i; on peut chercher à minimiser le plus grand retard $T_{max} = max\ T_i$, ou bien la somme des retards $\sum T_i$, ou encore la somme pondérée des retards $\sum w_i T_i$, voire le nombre de tâches en retard $\sum U_i$ ou le nombre pondéré de tâches en retard $\sum w_i U_i$...

Minimisation d'un coût

Par exemple, le critère $\sum_i w_i C_i$ permet d'estimer le coût des stocks d'encours; en effet la tâche (i) est présente dans l'atelier entre les instants r_i et C_i (même si elle n'est exécutée qu'entre t_i et C_i), et, donc les stocks dont elle a besoin doivent être disponibles entre ces deux dates; d'où un coût $\sum_i w_i\ (C_i - r_i)$ égale à une constante près à $\sum_i w_i C_i$.

3. MÉTHODES D'ORDONNANCEMENT

Dans ce paragraphe, nous présentons : le graphique de Gantt, les problèmes d'ordonnancement à contraintes potentielles, les méthodes sérielles et le traitement des problèmes d'ordonnancement à contraintes potentielles lorsque la durée des tâches est fonction linéaire de l'argent utilisé pour le financer.

3.1. Le graphique de Gantt

Le graphique de Gantt n'est pas une méthode pour résoudre les problèmes d'ordonnancement *mais seulement une méthode pour représenter une solution.*

Cette méthode très ancienne est excellente car très facile à lire même pour les profanes. La figure 1 rapporte le diagramme de Gantt d'un ordonnancement à cinq tâches (1), (2), (3), (4), (5), de durée $d_1=6$, $d_2=3$, $d_3=4$, $d_4=5$, $d_5=5$ et utilisant respectivement 4, 1, 3, 2, 3 unités de ressource 1 et 8, 7, 10, 10, 4 unités de ressource 2 lorsque les dates d'exécution des tâches sont 0, 3, 6, 8, 10.

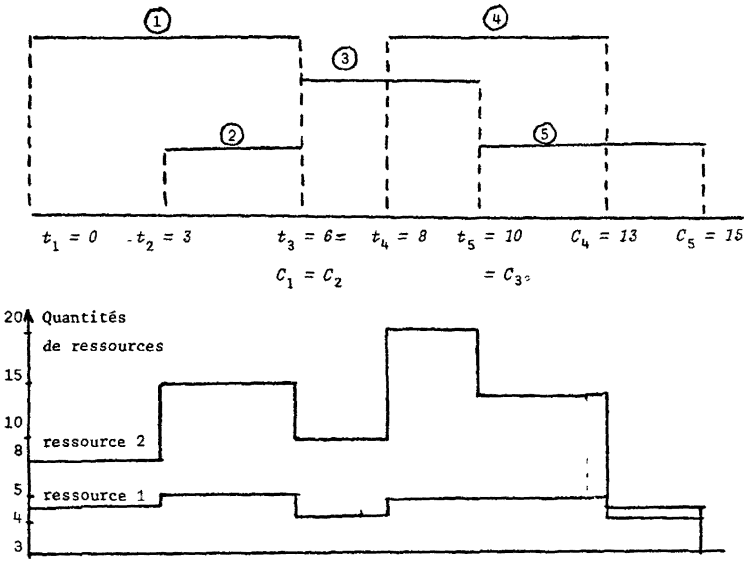


Figure 1. — Graphique de Gantt.

3.2. Traitement des problèmes d'ordonnancement à contraintes potentielles

Introduction

Ces méthodes permettent d'ordonnancer, en minimisant le délai d'exécution, un ensemble de tâches T soumises uniquement à des contraintes potentielles (et non à des contraintes de ressource) ([5]). Une application caractéristique est celle

([5]) Si la tâche (j) ne peut débuter que lorsque la tâche (i) est terminée, alors la contrainte potentielle associée s'écrit : $t_j \geqq t_i + p_i$ ou $t_j - t_i \geqq p_i$; plus généralement, une contrainte potentielle est une contrainte de la forme : $t_j - t_i \geqq a_{ij}$.

du bâtiment où l'entreprise constitue des stocks de matériaux avant d'entreprendre une construction : alors, seules les contraintes potentielles subsistent.

On distingue deux principales méthodes, la méthode C.P.M./P.E.R.T. ([6]) et la méthode des potentiels M.P.M., méthodes que nous allons illustrer à l'aide de l'exemple ci-dessous. Ces deux méthodes sont fondées sur la recherche d'un chemin de valeur maximale dans un graphe valué ([7]).

S'il est vrai comme le mentionnait vers 1948 un ouvrage consacré aux diagrammes de Gantt que : « le graphique de Gantt est la contribution la plus remarquable qui ait été apportée à l'art de la Direction au cours de cette génération » on pourrait sans doute, à l'instar de R. Faure, « tresser des couronnes d'or pur aux inventeurs de la méthode des potentiels et des diagrammes C.P.M. et P.E.R.T. ».

Exemple

On se propose d'ordonnancer sept tâches numérotées de 2 à 8, de durées respectives 3, 7, 4, 6, 5, 3, et 2; la tâche 2 précède les tâches 4 et 5; la tâche 3 précède la tâche 5; la tâche 4 précède les tâches 6 et 7; la tâche 5 précède la tâche 7; et la tâche 7 précède la tâche 8.

Quatre contraintes potentielles sont sous entendues : 2 et 3 peuvent débuter dès qu'on voudra, n'étant précédées par aucune autre tâche; 6 et 8, n'étant suivies par aucune autre tâche, précèdent l'achèvement général de l'ordonnancement.

Méthodes des potentiels

B. Roy a proposé de représenter le problème à l'aide d'un graphe $G=(X, U)$ ainsi constitué : l'ensemble de sommets X correspond à l'ensemble des tâches auquel on adjoint une tâche supplémentaire « début » numérotée (1) dont la date de début est : $t_1=0$ et une tâche supplémentaire « fin » numérotée (n); t_n est aussi la durée totale de l'ordonnancement; l'ensemble des arcs U est obtenu en associant à chaque contrainte potentielle (donc du type $t_j - t_i \geqq a_{ij}$) un arc (i, j) valué par a_{ij}; de plus l'arc $(1, i)$ figurera dans G valué par 0, si (i) est une tâche initiale; de même, si (j) est une tâche terminale, l'arc (j, n) figurera valué par $a_{jn}=p_j$.

([6]) A l'origine C.P.M. (Critical Path Method) désignait une méthode permettant de résoudre des problèmes pour lesquels les durées des tâches sont certaines; P.E.R.T. (Program Evaluation and Review Technic ou Program Evaluation Research Task), lorsque les durées sont aléatoires. En pratique ces deux sigles sont devenus synonymes, P.E.R.T. étant celui le plus fréquemment utilisé.

([7]) La valeur d'un chemin est la somme des valeurs des arcs de ce chemin.

Voici les principaux cas de valuations :

– (i) précède (j) : $a_{ij}=p_i$;

– (i) ne peut commencer avant r_i : $a_{1\,i}=r_i$;

– (j) peut commencer quand une fraction de (i) est achevée, par exemple, les deux tiers : $a_{ij}=2/3\ p_i$;

– (j) ne peut commencer que si (i) est terminée depuis au moins un temps θ : $a_{ij}=p_i+\theta$;

– (i) doit démarrer avant que θ unités de temps se soient écoulées depuis le début de (j) : $a_{ji}=-\theta$;

– la tâche (i) doit être achevée à la date d_i : $a_{i1}=p_i-d_i$; en effet $t_i+p_i \leqq d_i$ s'écrit $t_1-t_i \geqq p_i-d_i$ (car $t_1=0$), d'où un arc $(i, 1)$ valué par $a_{i1}=p_i-d_i$.

On a donc $t_j \geqq \max_{(i,j)\in U}(t_i+a_{ij})$; le problème revient à minimiser t_n; une solution consiste donc à prendre $t_j=\max_{(i,j)\in U}(t_i+a_{ij})$; on reconnaît le problème de la recherche d'un chemin de valeur maximale entre (1) et les autres sommets du graphe.

Ainsi les dates au plus tôt des tâches sont obtenues en cherchant les chemins de valeur maximale issues du sommet 1; ici $t_1=0$, $t_2=0$, $t_3=0$, $t_4=3$, $t_5=7$, $t_6=7$, $t_7=13$, $t_8=16$, $t_9=18$ (*fig.* 2).

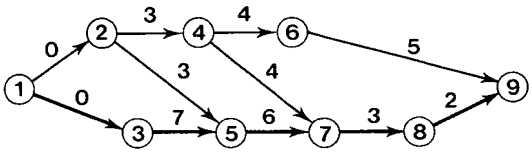


Figure 2. – Graphe potentiel tâches $G=(X, U)$.

Remarquons que :

1. Le problème a des solutions si et seulement si il n'existe pas de circuit « absorbant », c'est-à-dire de valeur strictement positive dans le graphe $G=(X, U)$ ([8]).

2. L'ordonnancement obtenu en considérant la valeur maximale t_i des chemins allant de (1) à (i) est une solution. Cet ordonnancement est dit « calé à gauche » car la tâche (i) ne peut pas commencer avant la date t_i dite alors date au plus tôt.

3. La durée minimale de l'ordonnancement est la valeur de ou des chemins de valeur maximale allant de (1) à (n); un tel chemin est dit « critique ». Si on

([8]) Une incohérence des données provoquerait l'apparition d'un circuit, par exemple, (2) précède (5) et (5) précède (2) introduirait le circuit 2, 5, 2.

retarde une tâche de ce chemin d'une durée δ, on retardera la fin de l'ordonnancement de la même durée (ou l'on rendra le problème impossible, en dépassant une date au plus tard d_k, ce qui se traduit dans G par l'existence d'un circuit absorbant passant par l'arc $(1, k)$.

De nombreux algorithmes permettent de déterminer l'ensemble des dates au plus tôt t_i : par exemple, si le graphe ne comporte pas de circuit (c'est le cas en général, si aucune date de début ou de fin au plus tard n'est imposée), l'algorithme le plus rapide est celui consistant à faire :

- $t_1 = 0$, marquer (1):
- puis choisir un sommet (j) dont tous les prédécesseurs (i) sont marqués et poser : $t_j = \max_{(i,j) \in U} (t_i + a_{ij})$;
- marquer (j);
- itérer tant qu'il reste des sommets non marqués.

La complexité théorique de cet algorithme est $O(m)$ où $m = Card\ U$. S'il y a des circuits la méthode matricielle (Roy, [36]) peut s'appliquer; elle a une complexité de $O(n^3)$.

Remarquons que le problème de la recherche d'un chemin de valeur maximale peut être formalisé à l'aide d'un programme linéaire dont les variables duales sont les potentiels (Wagner, [42]).

Méthode C.P.M./P.E.R.T.

Nous rappelons brièvement le principe de cette méthode (pour un exposé détaillé, *voir* par exemple [15]).

Dans la méthode C.P.M./P.E.R.T., on associe également au problème un graphe valué pour lequel on recherche les chemins de valeurs maximales; cette fois les sommets du graphe $H = (Y, V)$ sont des événements identifiés au début ou à la fin de tâche(s), et, les arc représentent, soit les tâches réelles, soit des tâches fictives permettant de représenter certaines contraintes potentielles (*fig.* 3).

On définit également l'ordonnancement calé à gauche et le(s) chemin(s) critique(s).

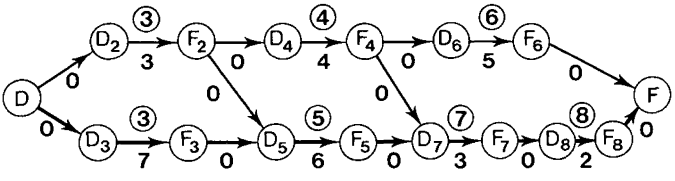


Figure 3. – Graphe P.E.R.T.. $H = (Y, V)$.

Voici comment peuvent être représentées dans le graphe P.E.R.T. les différentes contraintes considérées dans la méthode potentiel tâches :

– (i) précède (j), par un arc fictif (F_i, D_j), valué par 0:

– (i) ne peut commencer avant r_i, par un arc fictif (D, D_i), valué par r_i:

– (j) peut commencer quand les deux tiers de (i) sont achevés : on considère que la tâche (i) est l'union de deux tâches (i_1) et (i_2) où (i_1) représente les deux premiers tiers de (i) et (i_2) le dernier tiers; puis on introduit un arc fictif (F_{i_1}, D_{i_2});

– (j) ne peut commencer que si (i) est terminée depuis au moins un temps θ, par un arc fictif (F_i, D_j) valué par θ;

– (i) ne peut se terminer après d_i, par un arc fictif (F_i, D) valué par $-d_i$;

– (i) doit commencer avant la date α, par un arc fictif (D_i, D) valué par $-\alpha$.

Un inconvénient de la méthode C.P.M./P.E.R.T. par rapport à la méthode des potentiels est de doubler (initialement) les sommets du graphe car on associe à chaque tâche un événement « début de tâche » et un événement « fin de tâche »: ce graphe peut toutefois être simplifié par suppression des tâches fictives superflues et par fusion de certains événements : sur la figure 4, l'événement b résulte de la fusion des événements F_2, D_4, D_5; mais le défaut du graphe simplifié est sa rigidité vis-à-vis d'une modification de l'ensemble des contraintes (adjonction ou suppression).

D'autre part des fusions mal à propos introduisent des erreurs dans le graphe, sous forme de contraintes étrangères au problème; une règle permettant d'éviter ce type d'erreur est de fusionner des événements fin de tâches seulement si celle-ci ont les mêmes suivants; de même il convient de ne fusionner que des débuts de tâches ayant les mêmes antécédents.

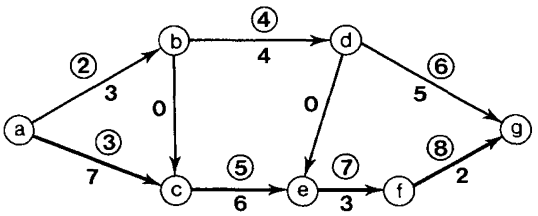


Figure 4. – Graphe C.P.M. simplifié.

Un avantage de cette représentation est qu'elle est plus facilement lisible par les praticiens car une tâche est représentée par un trait et un seul; de plus en disposant ces traits horizontalement et en leur donnant une longueur proportionnelle à la durée de la tâche, on présente le graphe P.E.R.T.. sous forme d'un diagramme de Gantt.

3 3. Méthodes sérielles

Jusqu'à présent, nous n'avons pas tenu compte des *ressources* : quand on fait apparaître les ressources (argent, machine, main d'œuvre) dont la disponibilité est limitée, la difficulté des problèmes est grande : nous verrons, dans la deuxième partie de l'article, qu'ils sont « *NP*-complets » (*voir* glossaire). Pourtant les problèmes se posent dans l'industrie et doivent être résolus; dans la pratique, on utilise des méthodes « sérielles » qu'il vaudrait mieux appeller algorithmes de liste.

Dans ces méthodes, une heuristique permet de classer les tâches selon un ordre de priorité; on ordonnance alors les tâches à partir de l'instant 0 en affectant à l'instant t parmi les tâches disponibles (c'est-à-dire celles dont les tâches précédentes sont achevées, et utilisant moins de ressources que les quantités disponibles à t) la tâche de plus haute priorité.

Un ordre de priorité pourra, par exemple, être un classement selon un ordre décroissant de dates au plus tard.

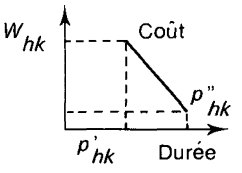
De nombreux programmes d'ordinateurs existent et sont disponibles pour traiter les problèmes d'ordonnancement compte tenu des ressources par diverses méthodes sérielles ou de lissage: par exemple, PROPLAN (Control Data): PROJACS (I.B.M.): PROMIS (Burroughs).

Notons que lorsqu'une solution est trouvée par une heuristique, certains essayent de l'améliorer, par exemple en étudiant les solutions voisines.

3.4. Accélération d'un ordonnancement en minimisant l'accroissement du coût

Présentation du problème

On s'intéresse, ici, à des problèmes comportant seulement des contraintes potentielles. Les contraintes d'antériorité sont représentées par un graphe P.E.R.T ; mais dans ce problème, la durée des tâches n'est pas fixée mais fonction linéaire décroissante de leur coût.



Toute tâche (i) est représentée par un couple (h, k) où ici h désigne l'indice de l'événement début de (i) et k l'indice de l'événement fin de (i).

A chaque tâche est donc associée sa durée minimale p'_{hk}, sa durée maximale p''_{hk}, et, l'augmentation w_{hk} de son coût lorsqu'on diminue sa durée d'une unité de temps dans l'intervalle (p'_{hk}, p''_{hk}).

Si la tâche (h, k) dure p_{hk} son coût sera donc :

$$W_{hk} - (p_{hk} - p'_{hk})\, w_{hk}.$$

Le problème consiste à trouver parmi les ordonnancements de durée λ un ordonnancement de coût global minimal (λ pouvant être éventuellement un paramètre).

Formulation du problème

Notons t_h la date de début d'exécution de l'événement h, p_{hk} la durée de la tâche (h, k). Le problème se formalise à l'aide du programme linéaire (P) :

$$(P) \qquad \begin{cases} t_k - t_h \geqq p_{hk} \text{ [pour toute tâche } (h, k)], \\ t_n \leqq \lambda, \\ p'_{hk} \leqq p_{hk} \leqq p''_{hk}, \\ Max \sum_{(h,k)} w_{hk} p_{hk}. \end{cases}$$

La première contrainte exprime que la tâche (h, k) a pour durée p_{hk}; la deuxième contrainte, que la durée totale de l'ordonnancement est inférieure à λ; les troisième et quatrième contraintes, que la durée p_{hk} de la tâche (h, k) est comprise entre p'_{hk} et p''_{hk}.

Pour traiter ce programme linéaire, Fulkerson a ramené par dualité ce problème à un problème de flot maximal à coût minimal. Pour une description détaillée *voir* [15].

4. PROBLÈMES A UNE MACHINE

Les résultats sur les problèmes d'ordonnancement sont trop nombreux pour pouvoir être tous décrits ici, c'est pourquoi, nous avons choisi de nous limiter à quelques problèmes particuliers en espérant donner une idée des méthodes employées, et des résultats que l'on peut établir. Le lecteur désirant une information exhaustive sur ce sujet pourra consulter [35].

4.1. Minimisation de la durée

A. Présentation

Le problème consiste à déterminer un ordonnancement de tâches sur une machine respectant des contraintes de dates au plus tôt et au plus tard. L'intérêt

de ce problème est accru par le fait que sa résolution permet de fournir une bonne évaluation par défaut de la durée minimale d'ordonnancement à contraintes disjonctives pour des problèmes à plusieurs machines distinctes.

Il faut fixer un ensemble de dates de début d'exécution $t_i (i \in T)$ tel que :

(i) deux tâches ne sont pas exécutées simultanément par la machine :

$$\forall (i, j) \in T \times T, \qquad t_j \geqq t_i + p_i \quad \text{ou} \quad t_i \geqq t_j + p_j;$$

(ii) l'exécution de la tâche i ne commence pas avant la date r_i :

$$\forall i \in T, \qquad t_i \geqq r_i;$$

(iii) la tâche i est achevée à la date d_i : $\forall i \in T$, $t_i + p_i \leqq d_i$.

S'il n'existe pas d'ordonnancement admissible, on abandonne la condition (iii) et on cherche un ordonnancement de retard minimal [le retard d'une tâche i est max $(0, d_i - (t_i + p_i))$]; le retard d'un ordonnancement est alors le retard de la tâche de plus grand retard.

B. Complexité

Le problème général est NP-complet : il est réductible au problème du partitionnement d'un ensemble E d'entiers en deux parties E_1 et E_2 de telle sorte que $\sum_{e_1 \in E_1} e_1 = \sum_{e_2 \in E_2} e_2$.

Le problème du partitionnement étant NP complet, il en est de même du problème à une machine (Lenstra, [30]).

C. Algorithmes approchés

Deux approches sont envisagées : proposer un algorithme approché ou mettre au point une méthode arborescente.

L'heuristique de Schrage ci-dessous permet de construire un ordonnancement dont le retard est faible sans toujours le minimiser. On rappelle que r_i désigne la date de début au plus tôt de la tâche i.

En cours d'algorithme, I désigne l'ensemble des tâches ordonnancées et, $\bar{I}$ celles non ordonnancées; t est le temps.

Heuristique de Schrage (Schrage, [40])

(i) Poser $t = \inf r_i$; $I = \emptyset$, $\bar{I} = T$.

(ii) A l'instant t, affecter, parmi les tâches i de $\bar{I}$ satisfaisant $r_i \leqq t$, celle de d_i minimal (ou l'une en cas d'égalité).

(iii) Poser $I = I \cup i$, $t_i = t$, $t = max\,(t_i + p_i, \underset{i \in \bar{I}}{inf}\, r_i)$; si $I = T$, fin; sinon retourner en (ii).

La nouvelle valeur de t définie dans (iii) représente l'instant auquel on utilisera de nouveau la machine pour une autre tâche : la tâche i mobilise la machine jusqu'à la date $t_i + p_i$ d'une part, et, d'autre part, les tâches non programmées ne peuvent être exécutées avant $\underset{i \in \bar{I}}{inf}\, r_i$.

L'algorithme de Schrage est optimal lorsque tous les r_i sont nuls; il s'agit alors de la règle bien connue de Jackson qui consiste à affecter les tâches selon leur urgence (plus petite date au plus tard).

Dans le cas général, le retard ne peut dépasser de plus de max p_i le retard minimal; l'exécution de l'algorithme requiert $O(n^2)$ opérations; une amélioration permet de n'avoir plus que $O(n \log_2(n))$ opérations.

D. Méthode arborescente

Pour bâtir une méthode arborescente, il faut une excellente évaluation par défaut de la durée optimale; cette évaluation peut être obtenue en autorisant le morcellement des tâches.

Dans le cas de tâches morcelables, le problème est polynômial et peut être traité, en modifiant légèrement l'algorithme de Schrage, en $O(n \log_2(n))$ opérations.

De nombreux auteurs ont proposé des méthodes arborescentes (Baker *et al.*, [3], Bratley *et al.*, [4], Lageweg *et al.*, [29]) utilisant l'algorithme de Schrage. Ces méthodes permettent de traiter des exemples de 1 000 tâches entre 1 et 10 secondes sur IRIS 80 (Carlier, [10]).

Remarquons que dans le cas où les durées sont égales, le problème est polynômial (Simons, [45]).

4.2. Minimisation du délai d'exécution pondéré

Par convention, dans la suite, lorsqu'il n'y a pas de relation de succession entre les tâches, nous dirons que les tâches sont indépendantes.

Nous considérons ici le critère :

$$Min\ \mathfrak{p} = \sum_{i=1}^{n} w_i C_i;$$

il s'agit d'ordonnancer des tâches soumises à des contraintes d'antériorité sans possibilité de morcellement. Ce problème est NP-complet dans le cas général; aussi allons-nous présenter quelques cas particuliers.

Nous allons voir que lorsque les tâches sont indépendantes, un argument d'échange entre tâches consécutives permet d'obtenir l'ordonnancement optimal. Lorsque les tâches sont liées par un ordre partiel représenté par un graphe G, on peut ramener le problème à la recherche de sous-ensembles dits « ρ-maximaux » (§ 4.2.2) qui conduisent à un algorithme polynômial (§ 4.2.3) dans le cas où G est une arborescence (Bruno *et al.*, [5] et [7]).

Enfin, nous exposons une extension de cet algorithme lorsque certaines données sont aléatoires.

D'autres cas particuliers ont reçu une solution polynômiale (Horn, [21]).

4.2.1. *Cas où les tâches sont indépendantes*

Considérons une solution du problème, c'est-à-dire une permutation $(i_1, i_2, \ldots, i_n)$ de T et voyons quel est l'effet d'une interversion de deux tâches consécutives i_k et i_{k+1}.

Notons φ' la valeur du critère pour la permutation obtenue. Nous avons alors (voir *fig.* 5) :

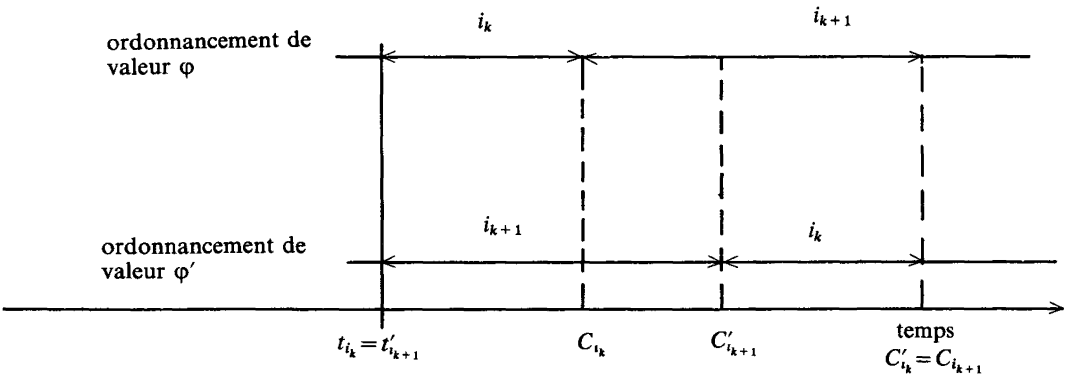


Figure 5

$$\varphi' - \varphi = w_{i_k}(C'_{i_k} - C_{i_k}) + w_{i_{k+1}}(C'_{i_{k+1}} - C_{i_{k+1}}) = w_{i_k} p_{i_{k+1}} - w_{i_{k+1}} p_{i_k}.$$

Il en résulte que :

$$\varphi' < \varphi \quad \Leftrightarrow \quad \frac{w_{i_k}}{p_{i_k}} < \frac{w_{i_{k+1}}}{p_{i_{k+1}}}.$$

Si l'on note : $\rho_i = w_i / p_i$ (indice de priorité de la tâche i) :

$$\varphi' < \varphi \quad \Leftrightarrow \quad \rho_{i_{k+1}} > \rho_{i_k}.$$

Nous constatons donc que, plus le taux ρ_i est grand, plus la tâche i est prioritaire. En l'absence de contraintes d'antériorité nous pouvons alors ordonner les tâches dans l'ordre des ρ_i non-croissants et obtenir ainsi une permutation (en général il en existe plusieurs) $(i_1^*, i_2^*, \ldots, i_n^*)$ telle que :

$$\rho_{i_1^*} \geqq \rho_{i_2^*} \geqq \ldots \geqq \rho_{i_n^*}. \tag{1}$$

Cette permutation est optimale car on peut s'y ramener à partir de toute solution par des échanges améliorants de type précédent.

4.2.2. *Cas où un ordre partiel est imposé*

Soit $U \subset T$, nous définissons par extension :

$$p(U) = \sum_{i \in U} p_i; \qquad w(U) = \sum_{i \in U} w_i$$

et $\rho(U) = w(U)/p(U)$ (indice de priorité de la partie U);
l'ordre partiel entre les tâches est représenté par un graphe $G = (T, A)$ où T est l'ensemble des tâches et A l'ensemble des couples (i, j) tels que i précède j.

Nous notons $G(U) = (T, A \cap U \times U)$ la restriction de l'ordre partiel à une partie U de T; s un ordonnancement des tâches de T, c'est-à-dire un ordre sur T compatible avec l'ordre partiel; $s(U)$ la restriction de s à U.

Exemple :

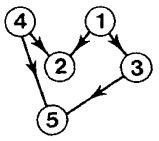


Figure 6

$$T = \{1, 2, 3, 4, 5\},$$
$$A = \{(1, 2); (1, 3); (4, 2); (3, 5); (4, 5)\},$$
$$s = (1, 4, 3, 2, 5).$$

Considérons $U = \{1, 2, 3\}$:

$$A \cap U \times U = \{(1, 2); (1, 3)\},$$
$$s(U) = (1, 3, 2).$$

Le diagramme de Hasse de l'ordre partiel est représenté sur la figure 6. On peut alors montrer la propriété suivante :

(P_1) : si s^* est une solution optimale pour G, et si $U \subset T$, alors $s^*(U)$ est une solution optimale pour G(U).

Nous dirons qu'une partie U est initiale si et seulement si :

(*a*) $U \neq \emptyset$; (*b*) $\omega^-(U) = \emptyset$ (i.e. tous les prédécesseurs de tâches de U sont dans U). Enfin, nous dirons qu'une partie U est ρ-maximale si et seulement si :

(1) U est initiale;

(2) $\rho(U)$ est maximal;

(3) U est minimale (pour l'inclusion) avec (1) et (2).

Les parties ρ-maximales sont alors propriétaires dans l'ordonnancement. En effet, on peut démontrer les 2 propriétés suivantes :

(P_2) : Si U est une partie ρ-maximale, il existe une solution optimale s telle que $s=(s(U), s(\overline{U}))$:

(P_3) : Si s est une solution optimale, il existe une partie ρ-maximale U telle que $s=(s(U), s(\overline{U}))$.

4.2.3. *Algorithme dans le cas où le diagramme de Hasse est une famille d'anti-arborescences* ([9])

Dans ce cas, chaque tâche a au plus un successeur. Notons H_i la partie constituée de i et de ses prédécesseurs. On peut alors montrer que l'une des parties H_i où $i \in T$, est ρ-maximale.

L'algorithme est alors construit comme suit :

(1) Détermination d'une tâche l telle que H_l soit ρ-maximale.

(2) D'après P_2, il existe une solution s optimale telle que $s=(s(H_l), s(\overline{H_l}))$; donc, d'après la structure de l'ordre partiel G on a $s=(H_l-\{l\}), l, s(\overline{H_l}))$, la tâche l est alors placée dans la permutation optimale.

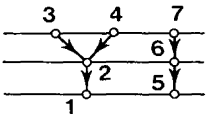
(3) En remarquant que $G(H_l-\{l\})$ et $G(\overline{H_l})$ sont aussi des familles d'antiarborescences, et grâce à P_1, on peut réitérer les points 1 et 2 pour $G(H_l-\{l\})$ et $G(\overline{H_l})$.

A chaque itération, une tâche supplémentaire est placée dans la permutation optimale. Le calcul des $\rho(H_l)$ nécessite $O(n)$ opérations lorsque l'on balaye les sommets des sommets pendants vers la racine. Par conséquent, la complexité de l'algorithme est $O(n^2)$ (Bruno *et al.* [6]).

Exemple : On considère l'ensemble des sept tâches ordonnées partiellement par le graphe de la figure 7 et auxquelles, sont attachées les données numériques

([9]) Une anti-arborescence a ses arcs orientés vers le puits; dans le cas d'une arborescence l'algorithme se généralise en remplaçant les ensembles initiaux par des ensembles terminaux. ($\omega^+(U)=\emptyset$).

suivantes :



i	1	2	3	4	5	6	7
p_i	1	2	1	3	1	2	2
w_i	6	4	3	3	6	4	3

Figure 7

Nous obtenons pour la première itération le tableau suivant :

i	1	2	3	4	5	6	7
H_i	1 2 3 4	2 3 4	3	4	5 6 7	6 7	7
$w(H_i)$	15	9	3	3	13	7	3
$p(H_i)$	7	6	1	3	5	4	2
$p(H_i)$	$2+\frac{1}{7}$	$1+\frac{1}{2}$	2	1	$2+\frac{3}{5}$	$1+\frac{3}{4}$	$1+\frac{1}{2}$

H_5 est une partie ρ-maximale; on peut alors placer la tâche 5. De plus, étant donné la structure de $G(H_5)$ les tâches 7, 6, 5 ne peuvent être exécutées que dans cet ordre dans la permutation optimale.

On a donc, pour l'itération suivante : $\overline{H}_5 = \{1, 2, 3, 4\}$; les valeurs du tableau sont ici toujours valables pour $G(\overline{H}_5)$ (en général, il n'en est pas ainsi), si bien que c'est maintenant H_1 qui est ρ-maximal.

Nous plaçons alors la tâche 1 qui sera la dernière tâche exécutée; dans le nouveau sous-graphe $G(H_1 - \{1\})$, $\{3\}$ est ρ-maximale, d'où la permutation optimale (7, 6, 5, 3, 4, 2, 1).

4.2.4. *Extension de l'algorithme au cas où les données sont aléatoires*

Définition du modèle

Tout programme informatique peut être considéré comme un graphe dont les sommets sont associés à des séquences d'instructions consécutives terminées par un test.

Une exécution est alors un chemin particulier de ce graphe. Dans ce modèle, nous nous restreignons au cas particulier où *le graphe associé à chaque programme est une arborescence.*

A chaque sommet, ou tâche, on associe un triplet $(P_i.\ W_i,\ S_i)$ de variables aléatoires où P_i est la durée de la tâche i. W_i un poids attaché à la tâche i. S_i le successeur immédiat de la tâche i. L'intérêt de ce modèle est qu'il s'agit de l'un des rares modèles aléatoires pour lequel on connaisse une stratégie optimale.

Plusieurs programmes ou jobs doivent être exécutés. Un système de jobs J est alors constitué :

– d'une famille de tâches T pour laquelle la relation d'antériorité notée G est une famille d'arborescences (chaque arborescence correspond à un job j);

– de n fonctions de répartition jointes F_i pour les triplets $(P_i.\ W_i.\ S_i)$.

Nous appellerons *stratégie* σ une application qui associe à chaque sous-système de jobs l'une de ses tâches initiales (soit $\sigma(J)$). Si l'on se fixe une stratégie σ_0, le délai d'exécution pondéré $\varphi(\sigma_0)$ est alors une fonction des seules variables aléatoires $(P_i.\ W_i,\ S_i)$: l'espérance mathématique de $\varphi(\sigma_0)$ soit $\overline{\varphi(\sigma_0)}$ est alors à minimiser sur l'espace des stratégies.

Exemple : Considérons le système de jobs de la figure 8 (pour $0<p<1$) :

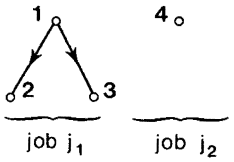


Figure 8

$$\text{Prob}\,[P_1=1.\ W_1=1/2,\ S_1=2]=p,$$
$$\text{Prob}\,[P_1=2.\ W_1=1.\ S_1=3]=1-p,$$
$$\text{Prob}\,[S_i=0]=1, \qquad i=2,\ 3,\ 4,$$
$$\text{Prob}\,[W_i=1]=1, \qquad i=2,\ 4,$$
$$\text{Prob}\,[P_i=1]=1, \qquad i=2,\ 3,\ 4,$$
$$\text{Prob}\,[W_3=5]=1.$$

Fixons-nous la stratégie σ_0 définie par :

$$\sigma_0(J)=1. \qquad \sigma_0(J/(2,\ 4))=4 \qquad \sigma_0(J/(3,\ 4))=3,$$

alors deux cas sont possibles :

1 suivi de 4 suivi de 2 (avec la probabilité p):

1 suivi de 3 suivi de 4 (avec la probabilité $1-p$), d'où l'espérance mathématique associée à σ_0 :

$$\overline{p(\sigma_0)} = p\left[1.\frac{1}{2} + (1+1)\,1 + (1+1+1)\,1\right]$$

$$+(1-p)\,[1.2+5(2+1)+1.(2+1+1)] = 21 - \frac{31}{2}p.$$

Deux exécutions sont possibles pour le job j_1 1 suivi de 2 ou 1 suivi de 3 avec les probabilités p et $1-p$.

Si l'on affecte la machine à la tâche 1 à l'instant $t=0$, il y a alors 2 possibilités à l'étape suivante :

(1) 1 se termine à $t=1$ et le système de jobs qui en résulte est $J' = J/\{2, 4\}$ noté δ $(J, 1, 2)$;

(2) 1 se termine à $t=2$ et le système résultant est $J/\{3, 4\}$ noté $\delta(J, 1, 3)$.

D'une manière plus générale, si J est un système de job, nous pourrons noter $\delta(J, i_0, i_1)$ le sous-système qui résulte de l'exécution de la tâche initiale i_0 d'un job j_0 de J et de la désignation aléatoire i_1 de la tâche suivante i_1 de i_0 dans j_0.

Définition de l'indice de priorité d'un job

Soit j_0 un job de J, U une partie initiale de $G(T(j_0))$ et i_0 la tâche initiale de j_0. A l'exécution de j_0 est associé un chemin d'extrémité initiale i_0 et dont l'extrémité terminale est une feuille de $G(T(j_0))$ [exemple (1, 2, 5) pour le job de la figure 9] ($T(j_0)$ est l'ensemble des tâches du job j_0).

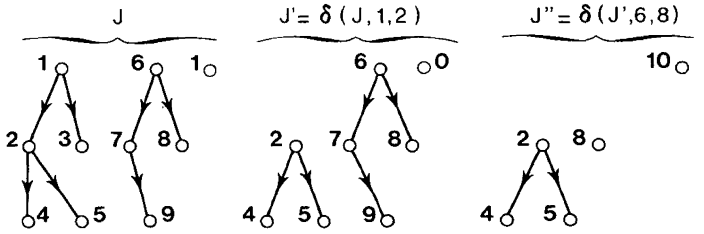


Figure 9

Soit $u(j, U)$ l'ensemble des tâches de ce chemin appartenant à U: le temps passé dans U est alors :

$$P(j, U) = \sum_{i \in u(j, U)} P_i \quad \text{(variable aléatoire)}$$

et le coût associé est :

$$W(j, U) = \sum_{i \in \mu(j, U)} W_i.$$

Nous notons alors $p(j, U)$ l'espérance mathématique de $P(j, U)$, et $w(j, U)$ celle de $W(j, U)$.

L'indice de priorité de la partie U de j est défini par :

$$\rho(j, U) = \frac{w(j, U)}{p(j, U)},$$

et l'indice de priorité du job j est alors :

$$\rho(j) = \max_{U} \rho(j, U), \qquad \text{où } U \text{ partie initiale de } j.$$

Une partie U sera ρ-maximale ($U \sqsubset j$) si :

(1) $U \neq \emptyset$:

(2) $\rho(j, U) = \rho(j)$:

(3) U est minimale [pour l'inclusion, avec les propriétés (1) et (2)].

Les propriétés suivantes conduisent à la détermination d'une stratégie optimale. Soit α_i la probabilité d'éxécuter la tâche i, on peut alors montrer que :

(1) $p(j, U) = \sum_{i \in U} \alpha_i p_i$ $(p_i = E[P_i])$,

$w(j, U) = \sum_{i \in U} \alpha_i w_i$ $(w_i = E[W_i])$.

(2) Si U est ρ-maximale et si la tâche $l (\in U)$ n'est pas initiale pour j, alors : $\rho(j/K_l) > \rho(j)$ (K_l est l'ensemble des tâches constitué de l et de ses descendants).

On est alors conduit au théorème suivant : une stratégie σ est optimale si et seulement si $\forall J, \sigma(J) =$ tâche initiale d'un job j^* tel que :

$$\forall j \in J : \rho(j^*) > \rho(j).$$

La machine est affectée au job dont le taux est le plus grand (ou à l'un de ceux-ci en cas d'égalité).

Exemple : Reprenons l'exemple de la figure 8 et fixons $p = 1/4$. Pour le job j_1, nous distinguons trois parties initiales :

$$U_1 = (1, 2, 3), \qquad U_2 = (1, 2), \qquad U_3 = (1, 3).$$

Pour le job j_2, une seule partie initiale $U_4 = (4)$.

Nous avons d'autre part : $\alpha_1 = 1$, $\alpha_4 = 1$, $\alpha_2 = 1/4$ et $\alpha_3 = 3/4$.

D'où d'après la propriété (1) :

$$p(j_1, U_1)=p_1+\frac{1}{4}p_2+\frac{3}{4}p_3; \qquad p(j_1, U_2)=p_1+\frac{1}{4}p_2;$$

$$p(j_1, U_3)=p_1+\frac{3}{4}p_3$$

et de même avec les « w ».

Or :

$$p_1=1.\frac{1}{4}+2.\frac{3}{4}=\frac{7}{4}; \qquad p_2=1; \qquad p_3=1;$$

$$w_1=\frac{1}{2}.\frac{1}{4}+1.\frac{3}{4}=\frac{7}{8} \qquad w_2=1; \qquad w_3=5.$$

Il en résulte que :

$$w(j_1, U_1)=1.\frac{7}{8}+\frac{1}{4}.1+\frac{3}{4}.5=\frac{39}{8},$$

$$p(j_1, U_1)=1.\frac{7}{4}+\frac{1}{4}.1+\frac{3}{4}.1=\frac{11}{4}$$

et :

$$\rho(j_1, U_1)=\frac{39}{8}\Big/\frac{11}{4}=\frac{39}{22},$$

$$w(j_1, U_2)=1.\frac{7}{8}+\frac{1}{4}.1=\frac{9}{8}, \qquad p(j_1, U_2)=1.\frac{7}{4}+\frac{1}{4}.1=2$$

et :

$$\rho(j_1, U_2)=\frac{9}{8}\Big/2=\frac{9}{16},$$

$$w(j_1, U_3)=1.\frac{7}{8}+\frac{3}{4}.5=\frac{37}{8}; \qquad p(j_1, U_3)=1.\frac{7}{4}+\frac{3}{4}.1=\frac{10}{4}=\frac{5}{2}$$

et :

$$\rho(j_1, U_3)=\frac{37}{8}\Big/\frac{5}{2}=\frac{37}{20}.$$

Il en résulte que :

$$\rho(j_1)=\max\left(\frac{39}{22},\frac{9}{16},\frac{37}{20}\right)=\frac{37}{20}.$$

De plus :

$$\rho(j_2)=\frac{w_4}{p_4}=\frac{1}{1}=1.$$

On a donc $\sigma^*(J)=1$; une fois la tâche 1 exécutée, 2 cas sont possibles :

1er *cas* : $\delta(J,1,2)=J'=\{j_1/(2),j_2\}$ les indices de priorité des deux jobs de J' sont aisément calculables (puisque réduits à une seule tâche).

$$\rho(j_1)=\frac{w_2}{p_2}=\frac{1}{1}=1 \qquad \text{et} \qquad \rho(j_2)=\frac{w_4}{p_4}=1,$$

donc nous aurons ici : $\sigma^*(J')=2$ ou 4.

2e *cas* :

$$\delta(J,1,3)=J''=\{j_1/(3),j_2\} \quad \rho(j_1/3)=\frac{w_3}{p_3}=\frac{5}{1}=5$$

et

$$\rho(j_2)=1,$$

donc $\sigma^*(J'')=3$.

Les systèmes résultant de J' ou J'' sont réduits à une tâche, dont la stratégie optimale σ^* affecte cette tâche au processeur.

4.3. Maximisation d'un profit (Lawler et Moore, [28], Sahni, [39])

A. Algorithme ε-approche

Un algorithme est dit ε-approché si, pour tout énoncé, la valeur $\hat{F}$ de la solution qu'il fournit et la valeur F^* de la solution optimale sont telles que :

$$|\hat{F}-F^*| \leq \varepsilon |F^*|$$

(i. e. : l'écart relatif entre $\hat{F}$ et F^* est inférieur à ε).

B. Introduction du problème

Dans le problème que nous considérons, toutes les tâches sont disponibles à $t=0$ (i. e. : $r_i=0$); l'achèvement d'une tâche i avant sa date de fin au plus tard d_i engendre un profit fixe w_i.

Notre but est de maximiser la somme des profits; donc, en notant $U_i = 1$ (resp. 0) si la tâche i est en retard (resp. à l'heure), la fonction objectif s'écrit :

$$max \sum_{i=1}^{n} w_i(1 - U_i).$$

Ce problème est NP-complet: Sahni a proposé pour le traiter des algorithmes ε-approchés. Nous en présentons deux faisant appel à une méthode de programmation dynamique: l'un utilise la troncature des données et a pour complexité $O(n^3/\varepsilon)$, l'autre la partition dynamique de l'intervalle des solutions et a pour complexité $O(n^2/\varepsilon)$.

Nous supposons que les n tâches sont numérotées dans un ordre croissant de leurs dates de fin au plus tard et qu'aucune tâche n'est d'emblée en retard ($p_i \leqq d_i$).

C. *Ordonnancements actifs*

Un ordonnancement optimal se trouve parmi les ordonnancements tels que :

– les tâches à l'heure ([10]) sont exécutées avant les tâches en retard:

– les tâches à l'heure sont exécutées dans l'ordre croissant de leurs dates de fin au plus tard.

En effet, si on recule les tâches en retard, la valeur de la fonction économique ne varie pas: il en est de même quand on transpose deux tâches à l'heure consécutives exécutées dans l'ordre décroissant de leurs dates de fin au plus tard.

Nous nous limitons à ces ordonnancements que nous appelons actifs et leur associons deux paramètres $\bar{w}$ et $\bar{p}$. $\bar{w}$ est le profit de l'ordonnancement considéré et $\bar{p}$ la somme des durées des tâches à l'heure.

Un ordonnancement actif est caractérisé par l'ensemble $I \subset T$ des tâches à l'heure; ainsi, nous pouvons confondre un ordonnancement actif avec l'élément J de $P(T)$ qui lui correspond: On a :

$$\bar{w} = \sum_{i \in I} \bar{w}_i, \qquad \bar{p} = \sum_{i \in J} p_i.$$

D. *Programmation dynamique*

On va calculer pour $i = 1$ à n des ensembles $O_i \subset P(I) (I = \{1, 2, \ldots, i\})$ d'ordonnancements sous-optimaux par rapport au critère de domination (*voir* plus bas). Un ordonnancement optimal sera dans O_n.

([10]) Une tâche est à l'heure si elle est achevée avant sa date au plus tard.

Critère de domination

Soient I_1 et I_2 deux parties de I dont les paramètres sont $(\bar{w}_1, \bar{p}_1)$ et $(\bar{w}_2, \bar{p}_2)$; I_2 *domine* I_1 si : $\bar{p}_2 \leqq \bar{p}_1$ et $\bar{w}_2 \geqq \bar{w}_1$.

Dans un ordonnancement optimal, la partie I' de I exécutée à l'heure n'est pas dominée par une autre partie de I; aussi peut-on se contenter de garder l'ensemble O_i des parties de I non dominées.

Introduction des S_i

Dans l'algorithme ci-dessous, afin d'avoir une méthode performante, on ne calculera qu'implicitement les ensembles O_i: S_i désignera un ensemble de couples $(\bar{w}. \bar{p})$, chaque couple étant associé avec un élément de O_i: S_i est ordonné dans le sens croissant des $\bar{w}$: l'ordonnancement optimal correspondra au plus grand couple de S_n.

Algorithme

Dans cet algorithme, on a comme :
données : $(w_i, p_i, d_i)\, 1 \leqq i \leqq n$ avec $d_1 \leqq d_2 \ldots \leqq d_n$ et comme,
résultat : le couple $(\bar{w}. \bar{p})$ d'un ordonnancement optimal reconstructible par remontée directe.

Étape 1 : *Initialisation :*

$$S_0 = \{0, 0\}.$$

Étape 2 : *Génération de* $S_1, S_2, \ldots, S_n$

Pour $i = 1$ à n, faire début :

$$V := \emptyset:$$

pour chaque couple $(\bar{p}. \bar{w})$ dans S_i, faire début
si :

$$\bar{p} + p_i \leqq d_i,$$

alors :

$$V := V \cup \{(\bar{w} + w_i, \bar{p} + p_i)\}:$$

fin :

Réunir S_{i-1} et V pour obtenir S_i; si au cours de cette opération, on rencontre $(\bar{w}_1, \bar{p}_1)$ et $(\bar{w}_2, \bar{p}_2) \in S_{i-1} \cup V$ tels que $\bar{p}_1 \geqq \bar{p}_2$ et $\bar{w}_1 \leqq \bar{w}_2$ alors éliminer $(\bar{w}_1, \bar{p}_1)$; fin.

Étape 3 : Détermination de la solution

Le profit associé à l'ordonnancement optimal correspond au dernier couple de S_n : cet ordonnancement est obtenu en remontant de S_n à S_0.

Pseudo-polynômialité de l'algorithme

Du fait de la procédure d'élimination, pour tout couple $(\bar{w}, \bar{p})$ et $(\bar{w}', \bar{p}')$ de S_i, on a : $\bar{w} \neq \bar{w}'$ et $\bar{p} \neq \bar{p}'$: en conséquence, il existe au plus :

$$M = min\left(\sum_{j=1}^{n} w_j, \sum_{j=1}^{n} p_j\right)$$

couples dans chaque S_i; la cardinalité de S_i est d'autre part majorée par 2^i. S_i étant calculable en $O(|S_i|)$ étapes, le temps requis pour calculer S_n est :

$$O\left(\sum_{i=1}^{n} |S_i|\right) = O\left(min\left(\sum_{i=1}^{n} 2^i, nM\right)\right) = O(min(2^n, nM)).$$

Cet algorithme est donc, si n est grand, en $O(nM)$, donc pseudo-polynômial: nous allons maintenant présenter deux algorithmes ε-approchés.

E. Troncature de w_i

Dans une première approche, nous tronquons chaque w_i en négligeant les d derniers chiffres.

Nous faisons alors une erreur d'au plus 10^d par w_i et donc d'au plus $n \times 10^d$ sur la fonction économique $\Sigma w_i(1-U_i)$; on a donc : $|\hat{F} - F^*| \leqq n.10^d$.

Remarquons que cet algorithme est alors :

$$\varepsilon = \frac{n\ 10^d}{w_{max}}$$

approché, car :

$$|\hat{F} - F^*| \leqq n.10^d = \varepsilon\, w_{max} \leqq \varepsilon\, F^* \text{ (car } w_{max} \leqq F^*).$$

Nous fixons maintenant ε, et calculons le nombre d de chiffres qu'il faut négliger pour avoir un algorithme ε-approché ; on a :

$$10^d \geqq \frac{\varepsilon w_{max}}{n} :$$

d'où :

$$d \geqq log_{10}\, \varepsilon + log_{10}\, w_{max} - log\ n.$$

La complexité de cet algorithme est :

$$O(n\,M) = O(n(nw_{max})) = O\left(n^2 \frac{w_{max}}{10^d}\right) = O\left(\frac{n^3}{\varepsilon}\right),$$

car la plus grande valeur des nouveaux w_i est :

$$w^*_{max} = \frac{w_{max}}{10^d}.$$

F. Partition dynamique de l'intervalle $[1, \Sigma\ w_i]$

Dans cette deuxième approche quand il y a deux couples dans S_i *avec des valeurs du profit très proches l'une de l'autre, on ne retient qu'un couple, celui qui a le plus petit* $\bar{p}$ (plus faible moyenne des durées).

Soit w la plus grande valeur du profit dans S_i (on calculera w avant d'éliminer les éléments appartenant à un même intervalle). On partitionne l'intervalle $[1, w]$ en $[n/\varepsilon]$ petits intervalles de taille $\varepsilon\, w/n$ et on retient pour chacun de ces intervalles, l'élément de S_i de plus petit $\bar{p}$.

Puisque F^* est plus grand ou égal à w, éliminer ces couples introduit une erreur d'au plus $\varepsilon\ w/n$ inférieure à $\varepsilon\, F^*/n$; les erreurs se cumulant, on a une erreur totale d'au plus $n.(\varepsilon\, F^*/n) = \varepsilon\, F^*$.

Le nombre d'éléments dans S_i que l'on conserve est au plus $O([n/\varepsilon])$; la complexité de cet algorithme est donc $O(n^2/\varepsilon)$; il est meilleur que l'algorithme précédent.

5. PROBLÈMES A m MACHINES

On peut les décrire comme suit :
un centre informatique dispose de plusieurs processeurs (certains peuvent être spécialisés, d'autres non); un ensemble de tâches liées par des contraintes doivent être exécutées quotidiennement (par exemple des programmes de maintenance), il faut alors déterminer un ordonnancement satisfaisant pour un ou plusieurs critères.

5.1. Problème non préemptif

On rappelle qu'un problème est « non préemptif » si l'exécution de chaque tâche ne peut être morcelée; m machines identiques doivent exécuter ces tâches en une durée minimale; il n'y a pas de relation d'antériorité entre tâches, ni de dates au plus tôt ou au plus tard; on suppose de plus que la durée des tâches est quelconque.

Le problème est *NP* complet dès que $m=2$ (Graham. [18] et [19]) : en effet il se ramène au problème du partitionnement d'un ensemble d'entiers en deux parties de même poids. Aussi des recherches se sont développées et proposent des méthodes heuristiques pour construire une solution. Nous notons C^*_{max} la durée minimale de l'ordonnancement et C_{max} la durée fournie par l'heuristique considérée. Graham a montré que :

$C_{max}/C^*_{max} \leqq 2-1/m$ pour tout algorithme de liste; rappelons qu'un algorithme de liste consiste à choisir un *ordre de priorité* sur les tâches (ou liste) puis, dès l'instant $t=0$, à allouer les processeurs libres à la tâche prête la plus prioritaire.

Graham a également montré. que pour la liste ordonnée selon les durées décroissantes :

$$C_{max}/C^*_{max} \leqq \frac{4}{3} - \frac{1}{3m}.$$

Coffman a obtenu une meilleure borne pour l'heuristique suivante : les tâches étant ordonnées par durées décroissantes, on fixe d'abord une durée maximale C_0 et l'on affecte les tâches dans l'ordre sur la première machine qui peut les exécuter, sans dépasser C_0. Si l'on ne peut placer toutes les tâches, on fera un nouvel essai avec $C_1 > C_0$, si on peut les placer, on fera un nouvel essai avec $C_1 < C_0$: si k essais ont été réalisés, on a : (Graham *et al.*. [20]) :

$$C_{max}/C^*_{max} \leqq 1,22 + \frac{1}{2^k},$$

pour une complexité de $O(n \log n + knm)$.

Remarquons que les trois bornes précédentes sont atteintes, c'est-à-dire correspondent au plus mauvais des cas, certains d'entre eux asymptotiques.

De nombreuses heuristiques ont été proposées par d'autres auteurs (Sahni, Ibara, Kim, Garey...).

Il apparaît que l'évaluation de leurs performances est mathématiquement très difficile.

5.2. **Problème préemptif** (Labetoulle *et al.*, [27]).

Cette fois, le morcellement est permis; une borne inférieure de la durée de l'ordonnancement est :

$$\max\left(\max_{i \in T}(p_i), \frac{1}{m}\sum_{i=1}^{n} p_i\right).$$

Mac Naughton (Coffman, [12]) construit un ordonnancement *optimal* en affectant les tâches aux machines, dans un ordre quelconque, une tâche étant éclatée chaque fois que cette borne est dépassée. On réalise ainsi également un nombre de préemptions au plus égal à $m-1$.

5.3. Problème préemptif avec relation d'antériorité

Quand le morcellement est permis et qu'il y a une relation d'antériorité, le problème est *NP* complet (Coffman, [12]). Toutefois, s'il y a deux machines ou si la relation d'antériorité est représentée par une anti-arborescence (cf. *fig.* 11), il existe des algorithmes pseudo-polynômiaux dont la complexité est un polynôme par rapport à n et à la durée maximale d'une tâche.

5.4. Problème non préemptif, à deux machines et durées égales

Deux algorithmes sont issus de remarques intuitives et correspondent à des ordonnancements construits à partir de listes de priorité. Malheureusement ces procédures ne sont plus suffisantes dès que la complexité (m croissant, les p_i distincts, graphe G quelconque) croît. Elles conduisent cependant à des heuristiques dont les performances sont difficilement appréciables.

Cas I: le graphe G est une anti-aborescence: les p_i sont égaux à une unité de temps: m est quelconque.

La notion qui paraît importante ici est celle de niveau d'un sommet, c'est à dire la longueur du plus long chemin allant du sommet au sommet terminal.

On construit une liste L ordonnée selon les niveaux non-croissants et l'on affecte les tâches aux machines selon L: *l'ordonnancement obtenu est optimal.*

La figure 11 rapporte les données d'un exemple ainsi que l'ordonnancement optimal sur trois machines pour $L=(14, 13, 12, \ldots, 1)$; sur la figure, entre $t=0$ et $t=1$ la tâche 14 est exécutée sur la machine 1, etc.

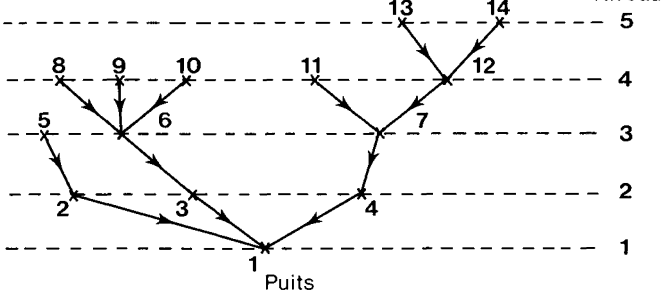


Figure 11

Machine \ Temps	1	2	3	4	5	6
1	14	12	8	6	3	1
2	13	10	7	4		
3	11	9	5	2		

Figure 11 *bis*

Notons A l'algorithme pour une liste L ordonnée selon les niveaux décroissants, l'optimalité de A peut être démontrée comme suit (Coffman, [12]).

On suppose A non optimal, il existe donc un *plus petit entier* n_0 et un système de tâches $S_0=(1, 2, \ldots, n_0)$ pour lequel A ne conduit pas à l'ordonnancement optimal, le délai étant alors f_0. On démontre alors les points suivants :

(1) il existe nécessairement une machine libre pendant l'intervalle de temps (f_0-3, f_0-2);

(2) on considère le système S_0' obtenu en supprimant la tâche terminale ainsi que ses prédécesseurs immédiats et l'on montre que A appliqué à S_0' donne l'ordonnancement obtenu en remplaçant les tâches supprimées par des périodes d'oisiveté; le délai est alors f_0-2;

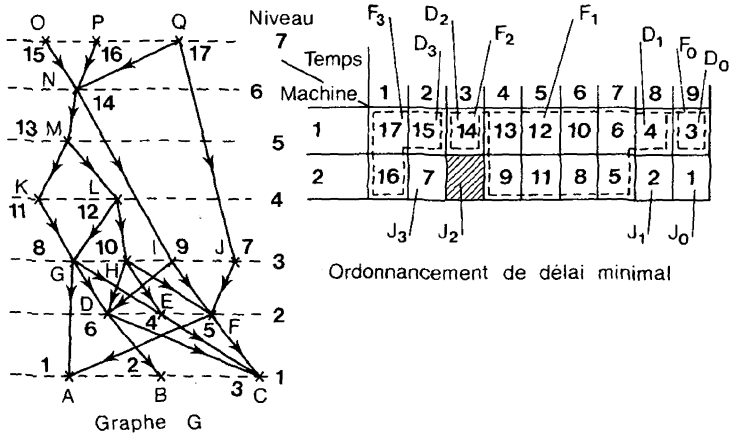


Figure 12. — Exemple.

(3) on transforme la forêt S_0' en arborescence S_0'' en créant une racine (tâche terminale); A appliqué à S_0'' fournit un ordonnancement de délai f_0-1; or le système S_0'' possède au plus n_0-1 tâches. Si on a réalisé les mêmes

transformations $S_0 \to S_0' \to S_0''$ sur le tableau de l'ordonnancement *optimal* associé à S_0, on obtient un ordonnancement réalisable pour S_0'' de délai $f_1 - 1$ (f_1 étant le délai *minimal* pour S_0); donc A ne serait pas optimal pour S_0'', d'où la contradiction.

Cas II : G est quelconque (représenté par son diagramme de Hasse, c'est-à-dire sans arc de transitivité) (Coffman *et al.*, [12]); les p_i sont égaux à une unité de temps; $m = 2$.

Ici aussi la notion de niveau est prépondérante; mais la difficulté vient du fait que des tâches de même niveau ne peuvent plus être considérées comme « équivalentes ».

Par exemple : Q est d'une priorité plus élevée que P puisque le seul successeur N de P est aussi successeur de Q; H est d'une priorité plus élevée que G car, outre D et E qui sont communs, le successeur F de H est de niveau supérieur à celui du successeur A de G.

Une tâche i sera d'une priorité plus élevée que celle de j si et seulement si :

(1) niveau de $i \geqq$ niveau de j (les niveaux sont comptés de bas en haut);

(2) en cas d'égalité, la liste des successeurs de i est lexicographiquement plus grande que celle de j.

Pour comparer deux listes, on ordonne chacune d'elles par priorités décroissantes.

On construit alors une liste L et l'ordonnancement issu de l'algorithme de liste correspondant (voir *fig.* 12); soit B cet algorithme.

Optimalité de B : elle est fondée sur la mise en évidence, dans l'ordonnancement associé à B, de sous-ensembles de tâches *réelles* (i. e. ne correspondant pas à des périodes d'oisiveté) $F_k, F_{k-1}, \ldots, F_0$ tels que toute tâche de F_i *précède* toute tâche de F_{i-1} (voir *fig.* 12). Pour cela, si lorsque les deux machines sont libres, on décide d'utiliser d'abord la machine 1, on va distinguer dans l'ordonnancement associé à B, certaines tâches (réelles ou non) qui sont exécutées *avant* des tâches de priorité plus élevée (par exemple la tâche 7). Ces tâches sont nécessairement exécutées par la machine 2. Sur notre exemple nous distinguerons la tâche 1 (par convention) dernière tâche réelle exécutée par la machine 2, nous noterons J_0 cette tâche et D_0 la tâche exécutée par la machine 1 pendant le même temps. J_1 sera alors la *dernière* tâche (réelle ou non) exécutée avant D_0 et de priorité plus faible que celle de D_0. On détermine ainsi (J_k, D_k) à partir de J_{k-1} et D_{k-1}. On obtient alors les parties F_k constituées :

- des tâches *réelles* exécutées après (J_k, D_k) et avant (J_{k-1}, D_{k-1});
- de la tâche D_{k-1} (*réelle* aussi).

Ces sous-ensembles F_k sont exécutés :

- consécutivement,
- de manière optimale (chacun).

L'ordonnancement construit par B est donc optimal.

Terminons par une remarque : pour m fixé >2, le problème est ouvert, en ce sens que l'on ne sait pas s'il est polynomial ou non

5.5. Problème préemptif à machines différentes (Labetoulle *et al.*, [26], Slowinsky, [41])

Cette fois il n'y a pas de relation de précédence mais les machines sont différentes. On note p_i^k le temps d'exécution de la tâche i par la machine k. La tâche i peut être exécutée partiellement par chaque machine. On résoud d'abord le programme linéaire suivant pour lequel $y = C_{max}$:

$$\text{MIN } y,$$

$$\sum_{k=1}^{m} x_i^k / p_i^k = 1 \qquad \text{pour} \quad i = 1, 2, \ldots, n,$$

$$\sum_{k=1}^{m} x_i^k - y \leqq 0 \qquad \text{pour} \quad i = 1, 2, \ldots, n,$$

$$\sum_{i=1}^{n} x_i^k - y \leqq 0 \qquad \text{pour} \quad k = 1, 2, \ldots, m,$$

$$x_i^k \geqq 0 \qquad \text{pour} \quad i = 1, 2, \ldots, n \quad \text{et} \quad k = 1, 2, \ldots, m,$$

où x_i^k représente le temps passé sur la machine k par tâche i.

Il faut ensuite « découper les x_i^k en morceaux » et les placer en sorte que plusieurs machines n'exécutent pas simultanément la tâche i. Pour cela, on construit une suite d'ordonnancements partiels réalisables tels qu'à chaque étape toute contrainte du problème réduit soit satisfaite.

5.6. Problème préemptif avec dates au plus tôt et au plus tard (Graham *et al.*, [20])

Le problème de l'existence d'un ordonnancement respectant des dates de début au plus tôt et au plus tard se ramène un problème de flots sur un réseau de transport.

5.7. Délai d'exécution pondéré (sans morcellement)

Nous supposons ici que :

- les tâches sont indépendantes (il n'y a pas de relation d'antériorité);

– leurs poids w_i sont égaux à 1, donc $\varphi = \sum_{i=1}^{n} C_i$.

Soit P la matrice des durées des n tâches sur les machines (notation p_i^k = durée de la tâche i sur la machine k).

Nous supposons $P > 0$.

Un ordonnancement est alors défini par une partition de T en m classes B_1, B_2, ..., B_m (certaines toutefois peuvent être vides) où les tâches de B_k sont celles exécutées par la machine k. Chaque classe non vide est ensuite totalement ordonnée.

Exemple : $m=3$, $n=8$:

$$P = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \end{array} \begin{array}{c} \begin{array}{cccccccc} (1) & (2) & (3) & (4) & (5) & (6) & (7) & (8) \end{array} \\ \begin{pmatrix} 2 & 3 & 1 & 4 & 6 & 5 & 2 & 3 \\ 1 & 4 & 1 & 5 & 5 & 4 & 3 & 4 \\ 3 & 2 & 3 & 2 & 3 & 5 & 1 & 2 \end{pmatrix} \end{array}$$

Un ordonnancement réalisable est constitué de $B_1 = (3, 5, 8)$, $B_2 = \emptyset$, $B_3 = (1, 2, 4, 6, 7)$ *et* $\Pi_1 = (3, 8, 5)$, $\Pi_2 = \emptyset$, $\Pi_3 = (7, 4, 2, 6, 1)$; la valeur du critère est alors :

$$\varphi = \underbrace{1 + (1+3) + (1+3+6)}_{\text{machine 1}} + \underbrace{0}_{\text{machine 2}}$$

$$+ \underbrace{1 + (1+2) + (1+2+2) + (1+2+2+5)}_{\text{machine 3}}$$

$$+ (1+2+2+5+3) = 47.$$

On peut réduire le problème à celui de *la recherche d'un flot maximal de coût minimal* de la manière suivante : si la tâche i est suivie de s tâches sur la machine k, sa durée contribue à la valeur du critère pour la quantité $(s+1)p_i^k$.

Si donc, nous notons $s(i)$ le nombre de successeurs de la tâche i sur la machine où elle est exécutée [que nous notons $m(i)$], la valeur associée du critère est alors :

$$\varphi = \sum_{i=1}^{n} (s(i)+1) p_i^{m(i)}.$$

On considère alors une matrice C à $n.m$ lignes et n colonnes définies par :

$$C_{li} = (s+1) p_i^k,$$

si $l = s.m + k$, $1 \leqq k \leqq m$.

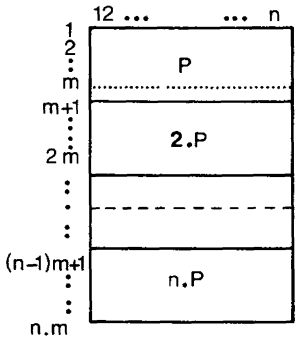


Figure 13

Cette matrice sera la matrice des coûts d'un problème de flot maximal de coût minimal; le réseau est représenté sur la figure 14. L'arc (l, i) de ce réseau est alors valué par C_{li}, la capacité des arcs est égale à 1 sauf celle de l'arc de retour égale à m.

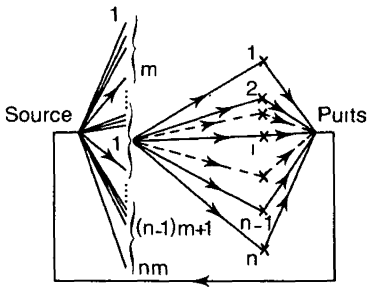


Figure 14

A un ordonnancement réalisable, il est aisé de faire correspondre un flot maximal; en effet si la tâche i est exécutée sur la machine k et admet $s-1$ suivants sur cette machine, on affecte l'arc (l, i) (où $l=sm+k$), ainsi que les arcs (source, l) et (i, puits) d'un flux unité.

Réciproquement, soit un flot maximal du coût minimal défini sur ce réseau, il lui correspond un ordonnancement réalisable, en effet si le flux de l'arc $(l, i)=1$ ($l=sm+k$), les sommets l' (où $l'=s'.m+k, s'<s$) sont saturés; dans le cas contraire nous pourrions :

(1) annuler le flux des arcs (source, l) et (l, i) et (i, puits);

(2) faire passer à 1 le flux des arcs (source, l') et (l', i) et (i, puits).

La variation de coût est alors :
$(s+1)p_i^k-(s'+1)p_i^k=(s-s')p_i^k>0$, ce qui contredit l'hypothèse d'optimalité du flot initial. En conclusion, la résolution du problème de flot maximal à coût minimal sur ce réseau permet de déterminer un ordonnancement optimal pour le critère du délai d'exécution pondéré.

6. PROBLÈMES D'ATELIER

6.1. Introduction

Les problèmes d'atelier apparaissent dans un contexte industriel. Un atelier contient m machines différentes et on veut fabriquer n pièces. On distingue dans la littérature différentes catégories de problèmes selon les contraintes imposées aux tâches :

(1) « FLOW-SHOP ».

L'ordre de passage des pièces sur les machines est imposé et est le même pour chaque pièce.

(2) « JOB-SHOP ».

Cet ordre reste imposé mais peut différer selon les pièces.

(3) « OPEN-SHOP ».

Aucun ordre n'est imposé.

Dans les problèmes d'atelier, une *tâche correspond au passage d'une pièce sur une machine.*

Dans la suite, nous considèrerons le problème du flow-shop.

6.2. Complexité

Le problème est NP complet dès que $m=3$ (Garey *et al.*, [17]). Il est résolu pour $m=2$ par l'algorithme de Johnson, les résultats restant valables lorsque le morcellement des tâches est permis.

6.3. Algorithme de Johnson [23]

Dans le cas où $m=2$, un job j devra passer un temps p_j^1 sur la machine 1, p_j^2 sur la machine 2, un ordonnancement optimal sera obtenu en adoptant la règle suivante.

Règle : Le job j devra précéder le job l sur les deux machines si :

$$\min(p_j^1, p_l^2) \leqq \min(p_j^2, p_l^1).$$

Cette règle définit un ordre optimal sur l'ensemble des tâches, son optimalité peut se démontrer par des arguments d'échanges de couples de tâches [15, 22].

6.4. Théorème de B. Roy [37]

Appelons O_i l'ordre de passage des pièces sur la machine i. Bernard Roy a montré que, pour le critère du délai total, il existe une solution optimale parmi les ordonnancements tels que :

$$O_1 = O_2; \qquad O_m = O_{m-1}.$$

Donc pour $m = 3$, il existe un ordonnancement optimal tel que $O_1 = O_2 = O_3$. Malheureusement, pour $m \geqq 3$, l'ordonnancement optimal ne fait pas forcément passer les pièces dans le même ordre sur les machines.

6.5. Méthodes arborescentes

Une évaluation de la durée optimale est obtenue en considérant les tâches exécutées par une machine et en abandonnant les autres contraintes de ressources. Le problème ainsi relâché est un problème à une machine.

Sur la base de cette évaluation et en utilisant des *relations de dominance* entre les ordonnancements, de nombreuses méthodes ont été proposées (Balas, [1]).

Certains recherchent la meilleure solution parmi les ordonnancements tels que : $O_1 = O_2 = \ldots = O_m$; cette restriction trouvant sa justification dans le théorème de Roy; des exemples à 50 pièces et 3 machines; 20 pièces et 5 machines ont été traités de manière optimale (Rinnoy Kan, [35]).

D'autres ne se restreignent pas; des exemples à 5 pièces et 12 machines, 6 pièces et 10 machines, 8 pièces et 9 machines ont été traités de manière optimale (Carlier, [8]); ces tailles étaient limites puisqu'un de ces exemples a demandé 10 minutes d'UNIVAC 1110.

6.6. Approches heuristiques

Les méthodes arborescentes ont deux inconvénients : d'une part les besoins de place en mémoire d'ordinateur risquent d'être très importants pour les grands problèmes; d'autre part, même pour les petits problèmes on n'est pas sûr d'obtenir l'optimum en un temps raisonnable.

Les méthodes heuristiques évitent ces deux inconvénients; elles permettent d'obtenir des solutions aux grands problèmes avec un effort de calcul limité et leur besoin de place en mémoire d'ordinateur est prévisible; bien sûr, elles ne garantissent pas l'optimalité et il est souvent difficile de juger de leur efficacité.

Une heuristique intéressante est celle proposée par Campbell, Dudek et Smith : à chaque étape, on résout un problème de Johnson de données $p_j'^1, p_j'^2$ pour construire une permutation. A la fin, on aura engendré $m-1$ ordonnancements parmi lesquels on retiendra le meilleur.

A l'étape 1 on pose : $p_j^{1j}=p_j^1$, $p_j'^2=p_j^m$; et à l'étape q :

$$p_j^{1j}=\sum_{k=1}^{q} p_j^k \qquad \text{et} \qquad p_j'^2=\sum_{k=1}^{q} p_j^{m-k+1}.$$

6.7. Problèmes d'atelier sans attente

Si l'on impose que toutes les tâches d'une même pièce s'exécutent sans interruption, le problème se ramène à celui du voyageur de commerce mais reste, bien sûr NP complet [2].

7. CONCLUSION

Formaliser un problème d'ordonnancement est facile; il est en général extrait d'une situation concrète, on envisage des tâches, des machines, des durées...

Ces problèmes font l'objet de multiples recherches. Les chercheurs ont travaillé essentiellement selon deux axes : proposer des *méthodes exactes* et des *méthodes approchées*.

Les méthodes exactes ont donné l'objet à des algorithmes polynômiaux, pseudo-polynômiaux, et des méthodes arborescentes; les méthodes approchées à des heuristiques, des méthodes arborescentes à seuil, des techniques d'amélioration d'une solution, et des méthodes ε-approchées.

Les outils utilisés sont ceux des chercheurs opérationnels : graphes et problèmes de cheminement, flots sur un graphe, programmation linéaire, programmation dynamique, méthodes arborescentes...

Les résultats sont nombreux mais souvent d'une grande complexité mathématique, même pour des propositions apparemment très simples.

Ils permettent de sérier la difficulté des problèmes réels mais aussi de proposer des heuristiques et des fonctions d'évaluation efficaces.

Il apparaît que le nombre de problèmes restant à résoudre dépasse de loin le nombre de problèmes résolus. Aussi reste-t-il beaucoup de travail pratique et théorique à faire !

ANNEXE

Nous reprenons les tableaux récapitulatifs du livre (Coffman *et al.*, [12]) donnant les principaux résultats concernant :

– la complexité des problèmes d'ordonnancement;

– les bornes associées aux heuristiques de liste utilisées pour la recherche de bonnes solutions. Certains résultats concernent le cas où des tâches ont besoin d'autres ressources que les machines pour être exécutées; si s ressources sont nécessaires, on note R_j^i la quantité de ressource $j(=1, 2, \ldots, s)$ exigée par la tâche i et m_j la disponibilité totale en ressource j.

Le premier tableau concerne les résultats relatifs au délai minimal (noté C^*_{max}).

m	p_i	$<$	Morcellement des tâches	Ressources s m_i	Complexité	Référence
–	Égaux	Arbo-rescence	Non	0	$O(n)$	[22]
2	Égaux	–	Non	0	$O(n^2)$	[11]
Fixé ($\geqq 3$)	Égaux	–	Non	0	Ouvert	
–	Égaux	–	Non	0	NP-complet	[42]
Fixé ($\geqq 2$)	1 ou 2	–	Non	0	NP-complet	[42]
Fixé ($\geqq 2$)	–	$\emptyset$	Non	0	NP-complet	[12]
–	–	Arbo-rescence	Oui	0	$O(n \log_2 n)$	[32]
2	–	–	Oui	0	$O(n^2)$	[33]
Fixé ($\geqq 3$)	–	–	Oui	0	Ouvert	

n	p_i	$<$	Morcellement	Ressources s	Ressources m_i	Complexité	Référence
–	–	$\emptyset$	Oui	0		NP-complet	[12]
2	Égaux	–	Non	–	–	$O(n^3)$	[12]
Fixé ($\geqq 2$)	Égaux	Arborescence	Non	1	–	NP-complet	[16]
Fixé ($\geqq 2$)	Égaux	–	Non	1	$(m_1 = 1)$	NP-complet	[12]
Fixé ($\geqq 3$)	Égaux	$\emptyset$	Non	1	–	NP-complet	[16]
2	Atelier p_i^k	Fixé	Non	0		$O(n \log_2(n))$	[23]
Fixé ($\geqq 3$)	Atelier	Fixé	Non	0		NP-complet	[17]
Fixé ($\geqq 2$)	Atelier	Quelconque	Non	0		NP-complet	[17]

Le deuxième tableau concerne le délai d'exécution pondéré, c'est-à-dire MIN $\sum_{i=1}^{n} w_i C_i$.

m	p_i	w_i	$<$	Complexité	Référence
–	(p_i^k)	Égaux	$\emptyset$	$O(n^3)$	[5]
–	(p_i^k)	Égaux	Famille de chemins	NP-complet	[6]
1	–	–	Arborescence	$O(n^2)$	[21]
1	–	–	–	Ouvert	
Fixé ($\geqq 2$)	–	–	$\emptyset$	NP-complet	[7]

Rappelons quelques notations utiles à la compréhension du troisième tableau, relatif aux performances des heuristiques de liste.

Nous notons :

C^*_{max}, durée minimale de l'ordonnancement;

$C^{(l)}_{max}$, durée de l'ordonnancement qui résulte de la liste l;

l_{CC}, liste des tâches ordonnées selon l'ordre *non croissant* des longueurs du chemin critique;

l_p, liste des tâches ordonnée selon l'ordre *non croissant* de leur durée;
l, liste de priorités quelconque.

Liste	<	Borne supérieure de C^*_{max}/C_{max}	Référence
l	–	$2-\frac{1}{m}$	[18]
l_{CC}	$\emptyset$	$\frac{4}{3}-\frac{1}{3m}$	[19]
l l_{CC}	$\emptyset$ Arborescence	$1+(m-1)\frac{max\, p_i}{\Sigma p_i}$	[19] [24]
l_{CC}	–	$2-\frac{1}{m}$	[12]

Glossaire

m nombre de machines,
n nombre de tâches,
T ensemble des tâches,
i indice d'une tâche ou d'un événement,
t_i *date de début d'exécution de la tâche* i ou de l'événement i,
C_i date de fin d'exécution de la tâche i,
P_i durée de la tâche i,
$p_{i,j}$ durée de la tâche (i, j) (graphe C.P.M./P.E.R.T.),
p_i^k durée de la tâche i sur la machine k,
d_i date de fin au plus tard de la tâche i,
r_i date de disponibilité de la tâche i,
T_i retard de la tâche i : $T_i = max\ (O,\ C_i - d_i)$,
U_i indice de retard : $U_i = 1$ si la tâche est en retard, 0 sinon,
w_i poids de la tâche i,
G ordre partiel sur l'ensemble des tâches,
$O(n^p)$ on dit qu'un algorithme est en $O(n^p)$ s'il existe n_0 et c tels que, pour n supérieur à n_0, son nombre d'opérations est majoré par cn^p.

Algorithme polynômial

Un algorithme est polynômial s'il existe un entier p tel que cet algorithme soit en $O(n^p)$.

NP-complet

Un problème est *NP*-complet si l'existence d'un algorithme polynômial pour le résoudre implique l'existence d'algorithmes polynômiaux pour toute une classe de problèmes difficiles tel que le problème du voyageur de commerce et la programmation linéaire en nombre entier...

Préemption

Morcellement d'une tâche au profit d'une tâche de priorité plus élevée.

REMERCIEMENTS

Nous remercions les professeurs Jean Abadie, Robert Faure et Bernard Lemaire pour les nombreux conseils qu'ils nous ont donnés lors de la rédaction de cet article.

BIBLIOGRAPHIE

1. E. BALAS, *Machine Sequencing via Disjunctive Graph: an Implicit Enumeration algorithm*, Operation Research, vol. 17, n° 6, 1969.
2. K. A. BAKER, *Introduction to Sequencing and Scheduling*, John Wiley and sons, 1974.
3. K. A. BAKER et Z. S. SU, *Sequencing with Due-Dates and Early Start-Times to Minimize Maximum Tardiness*, Naval Research Logistic Quarterly, vol. 21, mars 1974, p. 171-176.
4. P. BRATLEY, M. FLORIAN et P. ROBILLARD, *On Sequencing with Earliest Start and Due-dates with Applications to Calculations of Bounds for the* $(n, m, G/F_{max})$ *Problem*, Naval Research Logistic Quarterly, 1979, p. 20-57.
5. J. BRUNO, E. G. COFFMAN et R. SETHI, *Algorithms for Minimizing Mean Flow Time*, Proceedings, IFIPS Congress, North Holland Publishing Company août 1974, p. 504-510.
6. J. BRUNO et R. SETHI, *On the Complexity of Mean Flow Time Scheduling*, Technical Report, Computer Science Dept, Pennsylvania State University, 1975.
7. J. BRUNO, E. G. COFFMAN et R. SETHI, *Scheduling Independant Tasks to Reduce Mean Finishing Time*, Communications of the A.C.M., vol. 17, n° 7, 1974.
8. J. CARLIER, *Ordonnancements à contraintes disjonctives*, R.A.I.R.O., vol. 12, n° 4, novembre 1978.
9. J. CARLIER, *Financement et ordonnancements*, Rapport de recherche, IP n° 78-8, Paris-VI, 1978.
10. J. CARLIER, *Problème à une machine*, EURO-IV, Cambridge, 1980; Rapport IP n° 80-3; Paris-VI.
11. E. G. COFFMAN et R. L. GRAHAM, *Optimal Scheduling for two Processor systems, Acta Informatica*, vol. 1, n° 3, 1972, p. 200-213.
12. E. G. COFFMAN, *Computer and Job-Shop Scheduling Theory*, John Wiley and Sons, 1976.
13. R. W. CONWAY, W. L. MAXWELL et L. W. MILLER, *Theory of Scheduling*, Addison Wesley, Reading, Mass., 1967.

14. J. ERSCHLER, G. FONTAN et F. ROUBELLAT, *Potentiels sur un graphe non conjonctif et analyse d'un problème d'ordonnancement à moyens limités*, *R.A.I.R.O.*, vol. V-13, n° 4, 1979, p. 363.

15. R. FAURE, C. ROUCAIROL et P. TOLLA, *Chemins, Flots et ordonnancements*, Gauthier-Villars, 1976.

16. M. R. GAREY et D. S. JOHNSON, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, California, 1978.

17. M. R. GAREY, D. S. JOHNSON et R. SETHI, *The Complexity of Flow-Shop and Job-shop Scheduling*, Technical Report, n° 168, Computer Science Dept, Pennsylvania State University, 1975.

18. R. L. GRAHAM, *Bounds for Certain Multiprocessing Anomalies*, Bell Systems Technical Journal, vol. 45, 1966, p. 1563-1581.

19. R. L. GRAHAM, *Bounds for Certain Timing Anomalies*, S.I.A.M. Journal on Applied Mathematics, vol. 17, n° 2,1969, p. 416-419.

20. R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA et A. H. G RINNOY KAN, *Optimization and Approximation in Deterministic Sequencing*, Proceedings of DO 77, Vancouver.

21. W. A. HORN, *Single Machine Job Sequencing with Tree-Like Precedence Ordering and Linear Delay Penalties*, S.I.A.M. Journal on Applied Mathematics, vol. 23, 1972, p. 189-202.

22. T. C. HU, *Parallel Sequencing and Assembly Line Problems*, Operations Research,vol. 9, n° 6, 1961, p. 841-848.

23. S. M. JOHNSON, *Optimal Two and Three Stage Production Schedules*, Naval Research Logistic Quarterly, vol. 1, n° 1, 1954.

24. M. T. KAUFMAN, *An almost Optimal Algorithm for the Assembly Line Scheduling Problem*, I.E.E.E. Transactions on Computers, vol. TC-74, 1974, p. 1169-1174.

25. A. KAUFMAN et G. DESBAZEILLE, *La méthode du chemin critique*, Dunod, Paris, 1966.

26. J. LABETOULLE et E. L. LAWLER, *On Preemptive Scheduling of Unrelated Parallel Processors*, J.A.C.M. (à paraître).

27. J. LABETOULLE, E. L. LAWLER et A. H. G. RINNOY KAN, *Preemptive Scheduling of Uniform Machines Subject to Release dates* (à paraître).

28. E. L. LAWLER et J. M. MOORE, *A Functional Equation and its Application to Resource Allocation and Sequencing Problems*, Management Science, vol. 16, n° 1, septembre 1969, p. 77-84.

29. B. J. LAGEWEG, J. K. LESTRA et A. H. G. RINNOY KAN, *Minimizing Maximum Lateness on one Machine: Computational Experience and some Applications*, Statistica Neerlandica, vol. 30, 1976, p. 25-41.

30. J. K. LENSTRA *Sequencing by Enumerative Methods*, Mathematical Centre Tract, 69, Mathematical Centrum, Amsterdam, 1977.

31. S. LIN, *Computer Solutions to the Traveling Salesman Problem*, Bell Systems Technical Journal, vol. 44, n° 10, 1965, p. 2245-2269.

32. R. R. MUNTZ et E. G. COFFMAN, *Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems*, Journal de l'A.C.M., vol. 17, n° 2, 1970, p. 324-338.

33. R. R. MUNTZ et E. G. COFFMAN, *Optimal Preemptive Scheduling on Processor Systems*, I.E.E.E. Transactions on Computers, vol. C-18, n° 11, 1969, p. 1014-1020.

34. NEPOMIASTCHY, *Résolution d'un problème d'ordonnancement à ressources variables*, R.A.I.R.O., vol. V-12, n° 3, août 1978.

35. A. H. G. RINNOY-KAN, *Machine Scheduling Problem: Classification, Complexity and Computations*, Nyhoff, The Hague, 1976.

36. B. ROY, *Algèbre moderne et théorie des graphes*, Tome II, Paris, Dunod, 1970.
37. B. ROY, *Cheminements et connexité dans les graphes, applications aux problèmes d'ordonnancement*, revue METRA, vol. 62, série spéciale, n° 1, 1962.
38. B. ROY et M. C. PORTMANN, *Les problèmes d'ordonnancement, applications et méthodes*, Cahier du LAMSADE, Université Paris-9, 1979.
39. S. K. SAHNI, *Algorithms for Scheduling Independant Tasks*, Journal de l'A.C.M., vol. 23, n° 1, janvier 1976, p. 116-127.
40. L. SCHRAGE, *Solving Resource—Constrained Network Problems by Implicit Enumeration: Preemptive Case*, Operations Research, vol. 20, n° 3, 1972, p. 668-677.
41. R. SLOWINSKY, *Multiobjective Network Scheduling with Efficient Use of Renewable and Non-Renewable Resources*, European Journal of Operational Research, vol. 5, 1980.
42. J. D. ULLMAN, *Polynomial Complete Scheduling Problems*, Operating Systems Review, vol. 7, n° 4, 1973, p. 96-101.
43. H. WAGNER, *Principles of Operations Research with Applications to Managerial Decisions*, p. 185-187, Prentice-Hall, 1969.
44. J. WEGLARZ, *Project Scheduling with Continously Divisible, Doubly Constrained Resources*, Management Science, 1981, (à paraître).
45. B. SIMMONS, *A Fast Algorithm for Simple Processor Scheduling*, 19th annual symposium on Fondations of Computer Science, octobre 1978, p. 246-251.